REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704 0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	29 June 1995	Annual Technical 1 July 1994- 30 June 1995

4. TITLE AND SUBTITLE

KQML - Accessible, High-Performance, Massive Knowledge Bases, Annual Technical Report, Year One

5. FUNDING NUMBERS

Grant Number

N00014-94-1-0907

6. AUTHOR(S)

James Hendler
Joel Saltz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Maryland
Department of Computer Science
College Park, Maryland 20742-3255

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Ballston Tower One
800 North Quincy Street
Arlington, Virginia 22217-5660

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

ONR, NRL Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We have begun porting the SIMD Parka system to more generic MIMD machines. The system has been recoded in C and supported using runtime optimization packages developed in the high performance computing laboratory at Maryland. New "scanning" algorithms have been developed for inheritance and recognition inferences. These algorithms have been tested with both random networks and on a recoding of the ontology of the CYC knowledge base. Tests show that the new version is significantly faster than the SIMD system, and that it promises to scale well to knowledge bases orders of magnitude larger than CYC.

14. SUBJECT TERMS

Knowledge based systems, High performance computing, interoperability

15. NUMBER OF PAGES

12

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Annual Report, Year I: N00014-94-0907

PI: James Hendler Co-PI: Joel Saltz

University of Maryland, College Park

Contract Title: KQML-Accessible, High-Performance, Massive Knowledge Bases

Contract Number: N-00014-94-0907

Principal Investigators: James Hendler, Univ. of Maryland
Joel Saltz, Univ. of Maryland

ARPA Order: B399

1 Research Results

The ParalleL Understanding Systems (PLUS) Laboratory at the University of Maryland has been working for a number of years on parallel support for very large knowledge bases. The main system to come out of this laboratory was the PARKA system, which was the SIMD implementation of a frame-based AI KR language on the CM-2. The system was shown to be extremely fast, and to perform extremely well on very large knowledge bases (having on the order of 10^5 frames). Details of the Parka system and its implementation can be found in [2, 3].

There were two main problems with Parka in its SIMD form. First, there's the obvious problem that SIMD computers are currently less popular than MIMD platforms. Second, the ability to scale to much larger knowledge bases than what could be maintained in the memory of the CM2 (i.e. we needed at least one (virtual) processor per node) was constrained by the hardware. This has led us to a desire to reimplement Parka on a more conventional MIMD platform, also moving from *Lisp to C.

In this report, we describe current status of this reimplementation effort. In particular, the goal of this project is to reimplement PARKA for generic MIMD computers. To reach this goal, we use the CHAOS-Library developed by Dr. Saltz and his students at the High Performance Computing Software Systems Laboratory at the University of Maryland [7]. The language is being implemented in as portable a manner as possible so as to work on a number of different MIMD computers.

1.1 Overview of new work

The need for portability has influenced the choice of the environments to implement PARKA:

Compiler: The compiler used was a standard ANSI C one. This compiler exists on all of the computers we currently access, particularly including the IBM SP-2 and the CRAY T3D.

Communication Library: Our implementation is based on functionality provided by the Chaos run-time support package which provides scheduling libraries that have been ported to a wide range of MIMD supercomputers. (Work on the Parka project has actually led to a new scheduling approach which is being integrated into the CHAOS libraries[8]).

Currently, we have focused on the design and implementation of a very general system that can easily be ported to the different supercomputers. This means that currently, no special optimizations are done using the specific characteristics of any one machine. (Future work will include optimizing on several important architectures.)

1.1.1 The data structures of the KB

To use Chaos and other generic tools, it was necessary to reimplement the PARKA system to be based on arrays, rather than on the linked lists of the earlier *lisp implementation. There are three major structures used:

Frames: The primary data structure used in the new implementation is a *frame*. A frame is an array of a variable size. Each element of the array points to an element of the Network Array described below. This pointer defines the properties of the frame. If an element points to a "normal" frame then the element represents an ISA-link. If the pointer goes to a frame that is characterized to be a property, then the link defines an explicitly valued property of the frame. This semantical difference allows us to separate the links into two sub-arrays, one for ISAs and one for properties.

A *property frame* (property) has a different structure than a normal frame. The basic structure of a property frame is again an array. This array contains two sub-arrays of the same size. For each non-ISA property in every normal frame, there is a pointer to a property frame. In the property frame, there are two links which are created, each in one of the sub-arrays. The link created in the first sub-array points back to the frame for which the property was defined. The second link points to the value for the property of that frame. In addition, a property frame also contains the arrays of a nomral frame (see Figure 1). (This allows properties themselves to be frames with hierarchies and other properties, which was not allowed in the earlier SIMD system.)

Network-Array: The network array contains is an array which holds as subarrays all of the normal and property frames in the system. (This array can be quite large for very large KBs.)

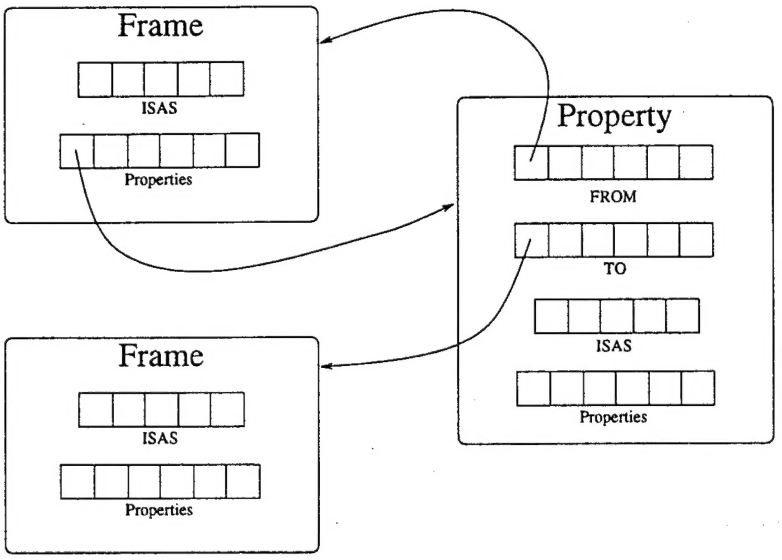

Figure 1: Frame Arrays in MIMD Parka

Frame-Table: The frame table is an array of strings. In each element of the frame table the name of a frame is stored. The name corresponds to the name of the frame located at the same position in the Network Array. (e.g. the fifth element of the frame table contains the name of the fifth element of the network). This allows a mapping from "human" to machine descriptors, for use in querying, etc.

This representation of the KB has several considerable advantages.

- The representation of the KB as an array (of arrays) allows us to use standard communication libraries like CHAOS. In addition, computation over array-based data structures is a well-studied area for MIMD Systems.
- The separation of the properties from the ISA links allows a fast parallel implementation for inheritance algorithms.
- The replacement of the single property link by a frame permits efficient implementations of different parallel algorithms such as recognition and structure matching.
- A separate name space (Frame Table) can be used to calculate starting points for scan operations without the presence of the whole KB.

1.2 Current Status

We have currently reimplemented three major portions of the PARKA system – the creation of frame-based semantic networks, the performance of inheritance operations thereon and the execution of recognition queries.

1.2.1 Construction of the Frames, the Network and the Frame Table

There are three methods for creating and/or loading the networks on a parallel machine.

1. The KB is defined inside of the program. There are commands like

   - define_frame
   - define_property
   - define_isa
   - assert

   that can be used to construct a KB.

2. A KB can be read in from an ASCII file containing these same commands.

3. If a KB is once constructed then the memory can be dumped on the disk. The dump files can be read in again later. It is possible to generate either a dump of the whole network or a distributed dump for multiprocessors. In the latter case, a separate dump for each node is created.

All three methods can be used to generate one network on a single node (processor) or a network being distributed over the nodes of the parallel computer. There are two predefined methods that distribute the network in blocks or stripes. (It is also possible to use other methods to distribute the network which are predefined or user-defined.)

In the first two methods (where a network is created) a consistency check for the whole network is done. For the third method it's assumed that the network is correct.

1.2.2 Inheritance

For the efficient performance of inheritance, we need an fast algorithm for scanning the ISA hierarchy. The efficiency of this algorithm is crucial to support for very large knowledge bases because, as we argue in [2], most interesting inferences in large knowledge bases are dependent on the efficiency (and correctness) of the inheritance algorithm.

The Scanning Algorithm The scanning algorithm is dependent on the distribution of the Network Array among the processors. The current implementation supports the two predefined types (block and cyclic) of distributions. The user is free to define other distributions, but all of these distributions must be static.

In principle, there are two approaches as to how the scan algorithm could work:

1. To expand the ISA hierarchy of a frame, a processor has to find out on which node the frame is located. If the frame is on the local node the ISA hierarchy can directly be expanded. If the frame is installed on a remote processor, then the local processor could ask for a copy of the frame from the remote processor and could then expand the hierarchy. This is essentially a "load balancing" solution, as frame information is propagated amongst processors.

2. The local processor has again to locate the frame in question and expand it directly if its local. Otherwise it will inform the remote processor to expand the corresponding frame. This approach (which we use) has more of a scheduling burden (coordinating communication) but needs less explicit load-balancing.

In the second approach, the work load is dependent on the distribution of the Network Array. Each frame is expanded on the processor which hosts that frame. The first approach does not contain such an implicit distribution of the work load. Instead, it would require that an explicit distribution of the work must be defined.

However, The amount of work to realize during the expansion of a frame is very small (only a few instructions) compared with the communication overhead. It would be very hard to implement an efficient load balancer because of the supplementary network load produced by the balancer. For this reason we chose the second approach for our implementation. Each frame is expanded by the processor on which it is stored. This leads to the following basic scan algorithm as used in this implementation:

1. `locate the starting frame`
2. `look up the ISA array of the frame`
3. `send msgs to all processors which host a frame defined in the ISA array`
4. `receive all msgs from remote processors`
5. `for each received frame goto step 2`

This program is executed in SPMD mode on each processor of the parallel computer. Based on this scan algorithm it's now possible to define an inheritance algorithm.

The Inheritance Algorithm The inheritance algorithm we use is based on the Touretzky IDO algorithm as modified in Parka. The theoretical worst case complexity of this algorithm is $O(d*n/k)$ where d is the depth of the ISA-hierarchy, n is the number of frames in the network and k is the number of nodes of the parallel computer.

The idea of our algorithm is that in one pass through the ISA-hierarchy, we can simultaneously create a list of all properties (direct and inherited) for each node and also a list of all properties "overwritten" by more specific properties (therefore being eliminated from the first list) The difference of these two lists contains the properties inherited according to IDO.

1.3 Potential Parallelism in KB

In this section we would like to analyze which kind/degree of MIMD-parallelism to exploit. The parallelism of a KB is hidden in the ontology. In our case the ontology is represented as a directed acyclic graph. In such a graph the parallelism usable by the scanning algorithm is defined by the branch out of each frame in the network (i.e. the number of ISA links per frame).

A network in which each frame has only one ISA link defined has no parallelism exploitable by the scanning algorithm. Each frame has to be completed before the descendent frame can be attacked. Such a graph imposes serial execution of the scanning algorithm. On the other hand, a graph with depth two and a large amount of ISAs defined for the root frame has the highest possible degree of parallelism for the scanning algorithm.

This algorithm therefore seems to be well adapted for AI KBs. Generally, these are not very deep but often very bushy. The fact that the parallelism is low for narrow ISA hierarchies is not very harmful. In addition, in several types of queries there are multiple scans to perform. These scans can be combined and executed as one single larger scan. Such a method is used to implement recognition queries (see Section 1.4) and in more complex inference methods like the structure matcher (see section 2.1).

1.4 Work in Progress

1.4.1 Recognition

We have also been implementing a new version of PARKA's *recognition* algorithm – its means for handling conjunctive property queries.[1] The algorithm is based on the special structure of the property frames and again uses the scanning algorithm. The complexity is also $O(d * n/k)$.

Is easy to find all explicitly labeled frames for a given property. The sub-array of the property frame with the back pointers contains all the pointers to the frames which are directly labeled for this property. The other sub-array of the property frame holds the pointers to the value for the property. On one hand it's possible to create a list with all the frames for which the property is directly defined and which have the searched value. On the other hand a list with all the frames which have not the searched value but which are able to "overwritten" the correct values during the inheritance can also be created.

The basic idea of the recognition algorithm is that all the frames which have the required property value are activated and start to send the informations downwards along the ISA links. During this scan, all frames in the ISA hierarchy will inherit the values of the property. If the inherited informations arrives on a node which is directly labeled for the property, but with a different value, then all the descendents from this node are commanded to remove the value they inherited from the same node as this node. Thus, it's possible to correctly handle all inherited properties according to IDO in one single scan.

The potential degree of parallelism of this algorithm is high, because there are multiple starting points (all the frames with direct correct property values). At the same time it's possible to send multiple properties across the ISA hierarchy during one scan step which results in a large scan graph with a higher degree of parallelism.

2 Testing

To realize the tests it was necessary first to program a network generator. The networks generated by this program were afterwards used to test the implemented KB.

[1]The SIMD algorithm for performing this was one of the main contributions of the original Parka system[4].

2.0.2 Network generator

The network generator is able to generate two kinds of KB formats:

1. ASCII file
2. memory dump files

There are three characteristics of the network that can be defined:

1. the branching factor in the network
2. the number of levels
3. the number of levels with the maximal size

Further it is possible to randomly generate properties for different frames. There is a parameter to define the density of these properties. Thus, this allows us to duplicate some of the tests made on the original Parka system on the SIMD CM-2 computer (cf. [2]).

2.0.3 Results on IBM SP2 (16 processors)

Degree of Parallelism As mentioned above, the performance of the scan algorithm depends on the structure of the ISA hierarchy. To test this, we created two networks, both with 2500 frames. The first network is 2500 levels deep and each level consists of one frame. The second network has two levels on the first level one frame is inserted and on the second 2499. For the sequential algorithm the amount of work is the same for both networks. This is also true for the parallel scanning algorithm but the big difference is that for the first network there is ooportunity for parallelism – one level has to be executed after another. In the second network it's different. All the frames on the second level can be treated in parallel if they are distributed among the processors.

To execute this first test, we used 4 processors on an IBM SP2. A cyclic network distribution was used. Thus, processor 1 owned the frames $\{0, 4, 8, ...\}$, processor 2 owned the frames $\{1, 5, 9, ...\}$, etc.

The execution of the scan algorithm on the first network took $0.4sec$. The scan through the second network only $0.008sec$. Note that although the theoretical speedup should be no more than 4, we have a speedup of 50! This is related to the communication system of the SP2 (and most other SIMD machines). For the first network we have to communicate 2499 short messages, in the second one long message. The throughput of the communication system is much higher on a small number of long messages then on a large number of short messages.

This result seems to very promising for real KB, because most of the existing KB haven a very flat ontology. For example, in a recoding of CYC for Parka (see section 2.0.3), the deepest subgraph we found was only 27 levels deep, whereas maximum branchout was over $1,000$.

nbr. nodes/netsize	1	2	4	8	16
29524	0.115	0.060	0.033	0.021	0.017
88573	0.345	0.175	0.091	0.050	0.032
265720	1.037	0.523	0.265	0.137	0.088
797161		1.689	0.735	0.388	0.201

Table 1: Inheritance

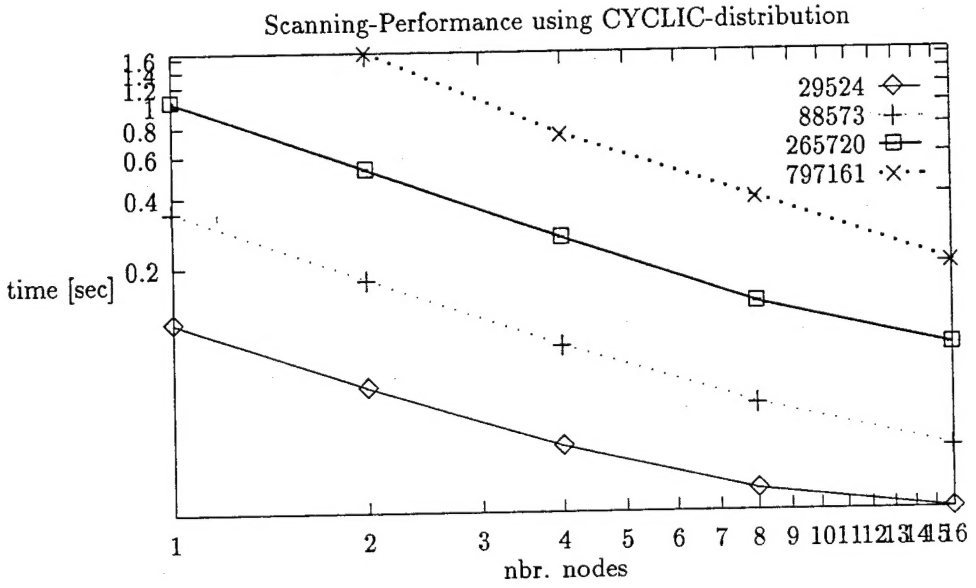

Figure 2: Scanning Performance

Scaling and Speedup As a second test, we want like to analyze how the scanning algorithm performs if the configuration of the Computer changes. The results presented here are again based on generated networks. The branching factor was fixed to 3. The depth of the ISA-hierarchy varies from 9 to 12, generating networks of about 29,500, 88,500, 265,700, and 797,000 frames respectively. (As a point of comparison, our encoding of the ontology of CYC (section 2.0.3) has about 32,500 frames.) A property to find was placed on the root node and the inheritance was done by one of the leaf nodes (this corresponds to the worst case for the previous Parka implementation). The absolute values represented in Figure 2 and table 1 are extremely fast, however we note that they are not so important because the programs were not optimized for the SP2 where the tests were done. What's interesting is the comportment of the algorithms. There are two points that seem to be interesting:

1. the scalability in terms of processor number

2. the scalability in terms of network size

On both measures, we see that the algorithms perform quite well. In addition, we believe further speedup would be available if we optimized the algorithms for this particular platform.

Results in CYC We have also realized some tests on KBs that were not randomly created. The largest one we used was the CYC ontology. As previously reported in [4], We encoded the ontology of version 8 of MCC's CYC system into Parka. In our version, this ontology has about 32,000 frames and about 150,000 assertions relating them (property and ISA links). The biggest difference between the generated networks and CYC is that the test network had only a single property on the root, thus making all "subgraphs" equal to the entire KB. In CYC the ISA hierarchies under particular properties are only subnetworks of the full 32000 frames.

These subnets are all much smaller than the smallest test network described in the previous section. For this reason simple queries like "What's the color of frames X" could not be used to show parallel effects – their time was on the order of 50-100μseconds on the single processor. Thus, instead of timing single inheritance queries, we looked for queries that would exercise more scanning.

To this end, we used queries of the following form: "Give me all the frames which have one or more properties in common with frame X". We also used recognition queries – conjunctive inheritances. Even with these more complex queries we had to chose either frames in big ISA hierarchies (like plant ot animal) or properties defined for a large number of frames (like color-of). The execution time for the queries "Give me all the frames with the same properties as Animal" and " Give me all the frames with the same property as Plan" are presented in figure 2.0.3

As one can see the recognition algorithm behaves very well. The efficiency is about 75%. That's very high in respect to the small amount of CPU time needed for the inheritance algorithm. The queries represented in the figure 2.0.3 are much more complex then standard queries in CYC. For recognition queries, we typically saw significantly faster times. For example, using the scan algorithms, we executed some recognitions with more then twenty conjuncts and had response times under 1/100 of a second (compared with about 1 second for the SIMD system).

2.1 Future work

2.1.1 Algorithms

Current work is focused on testing recognition and doing empirical research on how these algorithms scale to much larger networks and greater numbers of processors. To do this work, however, we must construct much bigger KBs then CYC. To this end we are pursuing two main areas of research – generating larger KBs by use of generative AI algorithms and developing hybrids of knowledge and databases (cf. [5]). We are also doing an analysis of the "topography" of CYC, so that we will be able to generate very large "CYC-like" random networks, allowing us to scale these algorithms to arbitrarily large networks. In addition, we are currently working on the reimplementation of structure matching inferences [1]. These

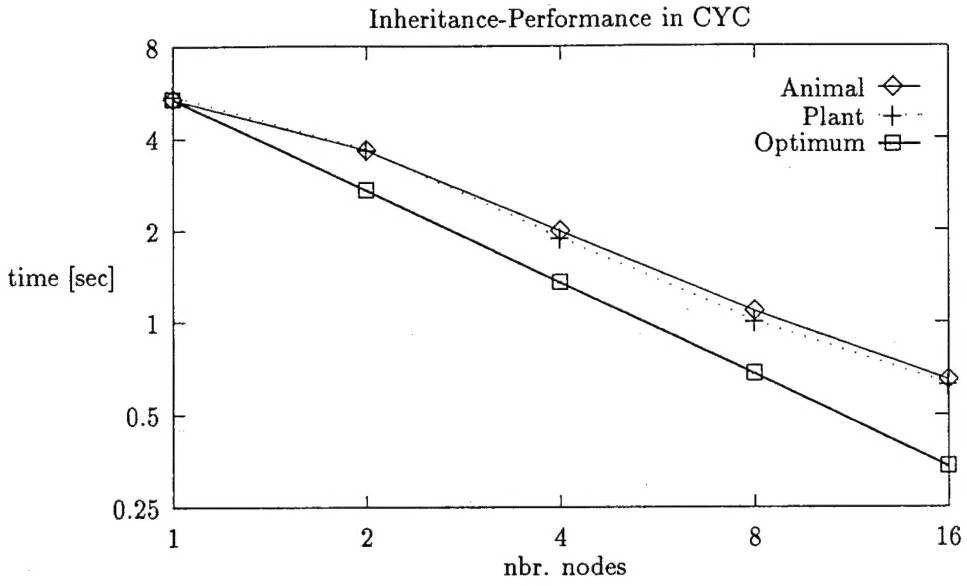

Figure 3: Recognition-Performance in CYC

inferences were crucial to the use of the PARKA system for case-based planning [6] and also form the basis of new work in creating browsers for hybrid knowledge/data bases.

2.1.2 Implementations

The current implementation is done on a small IBM SP2 machine (16 "wide" nodes). The next step will be to test these programs on bigger SP2s to see how the scaling continues in larger networks and on larger machines. We also will do work on optimizing communication patterns for the SP2 implementation (which should significantly improve absolute performance) as well as porting this implementation to the Cray T3D machine, which has significantly faster communication times. We expect to be able to run extremely large networks (potentially billions of frames) extremely efficiently on this machine. First tests of the basic scan algorithm seems to support this assumption.

In addition, we are exploring the use of I/O to allow the use of secondary storage for processing extremely large knowledge bases where even the online memory of the large parallel machines is not adequate. Chaos is being extended to support efficient I/O, and we are also beginning an examination of the special I/O requirements to storing and caching inferencing for scaling knowledge bases. The SP2 at Maryland is being configured to contain about 200Gigabytes of disk storage, making it uniquely suited for exploring massive knowledge and database applications.

References

[1] Andersen, W., Evett, M., Hendler, J. and Kettler, B. "Massively Parallel Matching of Knowledge

Structures," in *Massively Parallel Artificial Intelligence*, Kitano, H. and Hendler, J. (eds.), AAAI/MIT Press, 1994.

[2] Evett, M.P., Hendler, J.A., and Spector, L., "Parallel Knowledge Representation on the Connection Machine," *Journal of Parallel and Distributed Computing*, 1994.

[3] M.P. Evett. *PARKA: A System for Massively Parallel Knowledge Representation*, Ph.D. thesis, Dept. of Computer Science, University of Maryland, College Park, 1994.

[4] Evett, M.P., Hendler, J.A., and Andersen, W.A., "Massively Parallel Support for Computationally Effective Recognition Queries", *Proc Eleventh National Conference on Artificial Intelligence*, 1993.

[5] Hendler, J. High Performance Artificial Intelligence, *Science*, Vol. 265, August, 1994.

[6] Kettler, B.P., Hendler, J.A., Andersen, W.A., Evett, M.P., "Massively Parallel Support for Case-based Planning", *IEEE Expert*, Feb, 1994.

[7] Joel Saltz, Kathleen Crowley, Ravi Mirchandaney, and Harry Berryman. Run-time scheduling and execution of loops on message passing machines. *Journal of Parallel and Distributed Computing*, 8(4):303–312, April 1990.

[8] K. Stoffel, J. Hendler and J. Saltz, High Performance Support for Very Large Knowledge Bases, *Proc. Frontiers of Massively Parallel Computing*, Feb, 1995 (Extended Abstract).

3 Grant Related Activities

3.1 Awards And Honors

Professor Hendler received the Fulbright Foundation Fellowship for the 1995 Academic Year. He will spend 9 months at Hebrew University as part of this award.

3.2 Publications Resulting From Current Support

High Performance Artificial Intelligence, J. Hendler, *Science*, Vol 265, Aug 12, 1994.

Interprocedural Data Flow Based Optimizations for Compilation of Irregular Problems, Gagan Agrawal and Joel Saltz, *LCPC95.*

Performance Support for Very Large Knowledge Bases, K. Stoffel, J. Hendler and J. Saltz, High *Proc. Frontiers of Massively Parallel Computing*, Feb, 1995 (Extended Abstract).

Planning: What it is, What it could be, J. Hendler and D. McDermott, *Artificial Intelligence* (in press).

Runtime Support and Compilation Methods for User-Specified Data Distributions, R. Ponnusamy, J. Saltz, A. Choudhary, Y.-S. Hwang, and G. Fox, IEEE Transactions on Parallel and Distributed Systems. (in press)

Runtime Techniques for Parallelizing Sparse Matrix Applications, Manuel Ujaldon, Shamik D. Sharma, Joel Saltz, and Emilio Zapata, Proceedings of the Workshop on Irregular Problems 1995.

The Challenges of Real-Time AI, D. Musliner, J. Hendler, A. Agrawala, E. Durfee and J. Strosnider, *IEEE Computer*, 28(1), January, 1995.

Types of Planning — can artificial intelligence yield insights into prefrontal function?, J. Hendler, *The Frontal Lobes - Annals of the New York Academy of Science*, (in press).

The Case for Graph-Structured Representations, K. Sanders, B. Kettler and J. Hendler *Proceedings of the First International Conference on Case-based Reasoning*, New York: Springer-Verlag, 1995, in press.

A Critical Look at Critics in HTN Planning, K. Erol, J. Hendler and D. Nau, Proc. International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal (in press)

Formalizing Behavior-Based Planning for Nonholonomic Robots, V. Manikonda, J. Hendler and P.S. Krishnaprasad, Proc. International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal (in press).

A Motion Description Language and a Hybrid Architecture for Motion Planning with Nonholonomic Robots, V. Manikonda, P.S. Krishnaprasad and J. Hendler *Proc. International Conference on Robotics and Automation*, Nagoya, Japan, June, 1995.

Evaluating the CAPER Planning System, B.. Kettler and J. Hendler, *Proc. Israeli Symposium on Artificial Intelligence*, Jerusalem, Israel, Jan 1995.

Parka on MIMD-Supercomputers, K. Stoffel, J. Hendler and J. Saltz, *3rd International Workshop on Parallel Processing in AI*, Montreal, (in press)

Supporting Intelligent Real-Time Control: Dynamic Reaction on the Maruti Operating System, D. Musliner, R. Kohout and J. Hendler, *Proc. Israeli Symposium on Artificial Intelligence*, Jerusalem, Israel, Jan 1995.